Towards Correct Transformation: From High-Level Models to Time-Triggered Implementations

Hela Guesmi, Belgacem Ben Hedia, Simon Bliudze, Mathieu Jan, Saddek Bensalem

Towards Correct Transformation: From High-Level Models to Time-Triggered Implementations

Hela Guesmi*, Belgacem Ben Hedia*, Simon Bliudze†, Mathieu Jan* and Saddek Bensalem‡

*CEA, LIST, PC 172, 91191 Gif-sur-Yvette, France. Email: firstname.lastname@cea.fr
†EPFL IC IIF RiSD, Station 14, CH-1015 Lausanne, Switzerland. Email: simon.bliudze@epfl.ch
‡Verimag, 38610 Gieres, France. Email: Saddek.Bensalem@imag.fr

***Abstract*—In embedded systems, high-level component-based design approaches have been proposed in order to allow specification and design of complex real-time systems. However, their final implementations mostly rely on the generation of code for generic execution platforms. On the other hand, a variety of Real-Time Operating System (RTOS), in particular when based on the Time-Triggered (TT) paradigm, guarantee the temporal and behavioural determinism of the executed software. However, these TT-based RTOS do not provide high-level design frameworks enabling the scalable design of complex safety-critical real-time systems. The goal of our work is to couple a high-level component-based design approach based on the RT-BIP (Real-Time Behaviour-Interaction-Priority) framework with a safety-oriented real-time execution platform, implementing the TT approach. Thus, we combine their complementary advantages, by deriving correct-by-construction TT implementations from high-level componentised models. To this end, we propose an automatic transformation process from RT-BIP models into applications for the target platform based on the TT execution model. This transformation is already partially implemented.**

I. Introduction

The Time-Triggered (TT) paradigm for the design of real-time systems was introduced by Kopetz [11]. TT systems are based on a periodic clock synchronization in order to enable a TT communication and computation. Each subsystem of a TT architecture is isolated by a so-called *temporal firewall*. It consists of a shared memory element for unidirectional exchange of information between sender and receiver task components. It is the responsibility of the *TT communication system* to transport, by relying on the common global time, the information from the sender firewall to the receiver firewall. The strong isolation provided by the temporal firewall is key to ensuring the determinism of task execution and, thereby, allowing the implementation of efficient scheduling policies.

Developing embedded real-time systems based on the TT paradigm is a challenging task due to the increasing complexity of such systems and the necessity to manage, already in the programming model, the fine-grained temporal constraints and the low-level communication primitives imposed by the temporal firewall abstraction. Several Real-Time Operating Systems (RTOS) implement the TT execution model, such as for instance [3], [10]. However, they do not provide high-level programming models that would allow the developers to think on a higher level of abstraction and to tackle the complexity of large safety-critical real-time systems. Model-based design frameworks, such as [1], [7], allow the specification, the design and the simulation of real-time systems. In particular, the framework of [1] is a component-based framework for the design of real-time systems. It allows verification of behavioural properties, such as deadlock-freedom, and lends itself well to model transformations.

To the best of our knowledge, few connections however exist between high-level component-based design frameworksand TT execution platforms. A model transformation for generating distributed implementations from (non-real-time) BIP models is presented in [5]. A method for generating a mixed hardware/software system model for many-core platforms from a high-level non-real-time application model and a mapping between software and hardware components is presented in [8]. Nevertheless, these two approaches do not target the platforms based on TT execution model, thereby falling short of exploiting the strong temporal determinism guaranteed by the latter. A design framework based on UML diagrams and targeting the TT architecture is presented in [13]. [4] presents a transformation from SCADE [7] to PharOS [3]. The former does not target generic TT implementations since it assumes the underlying TT protocol to be the FlexRay standard, while the latter is limited to the relatively simple temporal behaviours. [6] presents a method to reduce the gap between models used for timing analysis and for TT code generation. Nevertheless, these approaches do not rely on a single semantic framework.

In this work, we establish a link between the model-based design framework RT-BIP [1] and a RTOS based on TT approach. Generating TT implementations from high-level RT-BIP models is achieved by a two-step transformation. The first step [9] transforms a generic RT-BIP model into a restricted one, which lends itself well to an implementation based on TT communication primitives. The second step, which is the subject of this paper, transforms the resulting model into the TT implementation provided by the PharOS RTOS. We identify the key difficulties in defining this transformation, propose solutions to address these difficulties and study how this transformation can be proven to be semantics-preserving.

This paper is structured as follows. In Section II, we provide the necessary background on RT-BIP and PharOS. The transformation is presented in Section III, while the open issues that remain to be addressed are discussed in Section IV.

II. Background

A. The RT-BIP Component Framework

RT-BIP is a component framework for constructing systems by superposing three layers of modelling: Behaviour, Interaction, and Priority. The Behaviour layer consists of a set of components represented by timed automata extended by data and functions given in C. The Interaction layer describes possible interactions between atomic components. Interactions are sets of ports allowing synchronizations between components. The third layer includes priorities between interactions using mechanisms for conflict resolution. Thus, in RT-BIP, systems are built by composing *atomic components* with *interactions* (presented by *connectors*) and *priorities*.

A *component* in RT-BIP is essentially a timed automaton [2] labelled by ports that represent the component's interface for communication with other components. A transition in RT-BIP automata can be constrained by a guard , i.e., a predicate on a set of its variables. A transition can also be constrained by timing constraint tc which is a guard over a set of clocks. Timing constraint is used to specify when actions of a system are enabled regarding system clocks. If c is a clock, a timing constraint tc over c is of the form: $l_c \leq c \leq u_c$, where $l_c, u_c \in \mathbb{R}_+$. Furthermore, in RT-BIP automata, a state l can be constrained by a time progress conditions (tpc) used to specify whether time can progress at a given state of the system. Any time progress condition tpc can be written as: $tpc = c \leq u_c$, where $u_c \in \mathbb{R}_+ \cup \{+\infty\}$. In the example of Figure 1, we display two RT-BIP components $C1$ and $C2$, composed by a binary connector. Let us assume that the system reaches the state $L1$ of $C1$ with a tpc equals to $c \leq 2$ and $c \in [1, 2]$. It can then either let the time progress until $c = 2$, or execute the transition enabled for these instants. If the state $L1$ is reached when $c = 2$, the system can not let time progress. It has to execute the transition p_1.

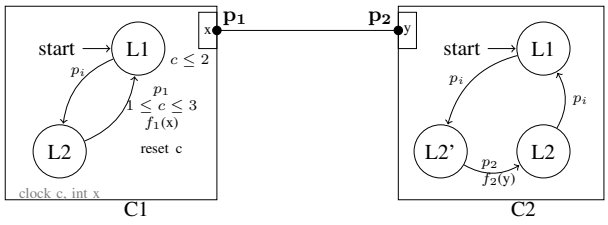

Fig. 1. Example of RT-BIP automata

B. TCA computation model and PharOS platform

Time-Constrained Automata (TCA) [12] is a formal computation model of TT tasks. The temporal behaviour of a task is specified using a directed graph, where arcs represent the successive jobs of the task to be executed (one at a time), and the nodes bear the temporal constraints of the jobs. There are four kinds of nodes:

- After node (*after(d)*): defines d as the relative release date of the following job. It is symbolized by $\triangleright_d$;
- Before node (*before(d)*): defines d as the relative deadline of the preceding job. It is symbolized by $\triangleleft_d$;
- Advance node (*advance(d)*): is a combination of $\triangleright_d$ and $\triangleleft_d$ nodes. It is symbolized by $\diamond_d$ and defines the absolute visibility date of the job data;
- No constraint node: imposes no temporal constraints on preceding and following jobs. It is symbolized by $\circ_d$.

A job can consult data whose absolute visibility dates are less or equal than the absolute release date of the job. The execution of an application can be seen as cyclically walking in the graph of each task and let the underlying scheduler choose when each encountered job is actually executed in the time interval defined by its release date and its deadline.

In the TCA example displayed in Figure 2, we have six jobs, labelled a to f, and five nodes. The release date of job d is one unit of time after the previous advance node. Two units of time after, job d should have ended. After jobs e and a are executed, communications take place since an advance node is used. The visibility date of data produced by e is three units of time later the previous after node.

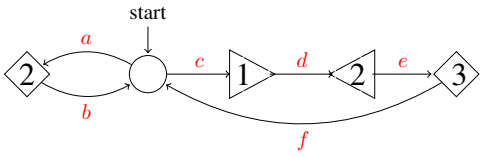

Fig. 2. Example of TCA automata

TCA computation model is implemented in PharOS [3]. PharOS is a method to design, implement and execute safety-critical multitasking applications based on the time-triggered paradigm. PharOS implements different variants of TT communication mechanisms. We are specially interested in the *temporal variables* which are real-time data flows. Values, available to all agents, are stored and updated by a single writer –the owner agent– at a predetermined temporal rhythm.

III. Work-in-progress: From RT-BIP to PharOS

In order to derive TT implementation from a high-level RT-BIP model, we follow a two-step transformation (see Figure 3).

Step1: RT-BIP model adaptation. This transformation consists in adapting the initial model, in order to comply with the TT paradigm, and especially the TT communication pattern. Each intertask interaction –initially held by connectors– is transformed in order to be handled by a dedicated component, standing for a medium between communicating tasks. The obtained model consists only of atomic RT-BIP components and connectors allowing unidirectional data transfer. Ports in TT-BIP model are send, receive or internal ports. This transformation is published in a previous work [9], and is proven to be semantics preserving.

Step 2: RT-BIP to TCA transformation. In this step, the output of the previous step is transformed into TCA automaton. In this section, we first detail challenges of the second step transformation. Then, we present the actual algorithm and how we are going to tackle the correctness proof part.

Fig. 3. From RT-BIP to the TCA TT computation model: a 2-step transformation.

A. Transformation subtleties

Transforming a component-based high-level model into a RTOS based system requires to address several subtleties.

Timing constraints mapping subtlety. The initial TT-BIP model is based on an abstract notion of time. In particular, it assumes that actions, corresponding to the computational steps of the system, are atomic and have zero execution times. Only start instant of these actions have timing constraints (tc) and timing progress conditions (tpc). However in TCA models, both release date and deadline of actions can be specified.

This issue is addressed by making use of the tpc notion in TT-BIP model, in order to extract the deadline of the following action. In fact the semantics of the tpc (used to specify whether time can progress at a given state of the system) and the *before* node in TCA are strictly similar. Both of them constraint the action preceding the node to finish before a certain date. The action succeeding the node starts right after the previous one. Timing constraint $l_c \leq c \leq u_c$ in TT-BIP, constrains only the start instant of the transition. In order to keep the same semantics in TCA, an empty action can be executed between nodes *after*(l_c) and *before*(u_c). Right after, the actions of the initial transition can be executed.

From absolute to relative constraints subtlety. In TT-BIP, all constraints are defined using absolute labelling. However, TCA nodes bear only relatively expressed constraints, i.e., as an increment to the previous $\triangleright_d$ or $\diamond_d$ node.

In order to address this second issue, we make use of the variable d_{ref}. It is initiated to zero and updated whenever a *before* node is instantiated. It stores then the current local clock value. Relative constraint ($d_{relative}$) is computed from its corresponding absolute constraint ($d_{absolute}$) following this formula:

$$d_{relative} = d_{absolute} - d_{ref} \quad (1)$$

Communication mapping subtlety. In the initial TT-BIP model, all tasks are related to communication components via connectors. Connectors provide not only unidirectional data transfer but also synchronization between sending and receiving actions of respectively the sender and the receiver components. In TCA, one communication model is called *temporal variable*. New values of temporal variables are made visible by their owners, i.e senders, at each of their synchronization points. Receivers of these data can consult their new values when their current time is equal or higher to the timing of these synchronization points. In our transformation two requirements need to be satisfied; (1) the receive must consult an updated temporal variable (i.e., after the sending action of the sender task) and (2) we need to respect communication semantics of the initial model.

This subtlety is addressed by generating TCA synchronization points (advance nodes) that depends on whether the TT-BIP transition is triggered by a send, receive or an internal port. After each nodes that corresponds to a communicating transition we instantiate an *advance(n)* node defined over a fine-grained clock. For example let a sender and receiver components having the same clock, and suppose they are meant to communicate in the same instant t in TT-BIP model. We can define a smaller clock, allowing to instantiate advance nodes (send and receive) at $t + \epsilon$. If we take the example of time lines displayed in Figure 4, the visibility instant of the sender data is $4*t+1$ of the clock g. The receiver will consult these data in the instant $4 * t + 2$ of the clock g.

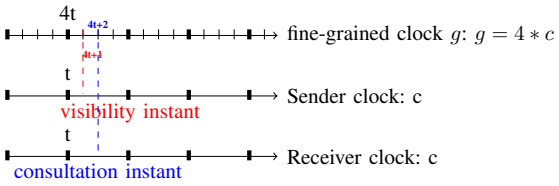

Fig. 4. Example of advance nodes defines on a fine-grained clock.

B. Transformation Algorithm

In Algorithm 1, for each transition of each automaton in TT-BIP model, we check if the transition source state has a tpc constraint. If this constraint exists, we instantiate a *before* node defined on the relative bound instant of this tpc. This before node defines the deadline of the previous action. The out transition of that node executes actions of the initial TT-BIP transition. If the transition source state tpc does not exist, and a timing constraint is defined over the transition, we instantiate two consecutive *after* and *before* nodes, defined successively over the lower and the upper bounds of the timing constraint of the initial transition. The transition relating these two states executes no actions. And then, we instantiate a transition to

Algorithm 1 Translation algorithm: Step 1

```
d_ref = 0;
for t=transition do
    if t.source.hasTpc then
        newTcaNode(before(t.source.tpc.value - d_ref));
        Synchro(t.labelPort);
        newOutTransition(t.actions);
    else if t.hasConstraint then
        newTcaNode(after(t.constraint.lowBound - d_ref));
        newOutTransition(update(d_ref));
        newTcaNode(before(t.constraint.upBound - d_ref));
        Synchro(t.labelPort);
        newOutTransition(t.actions);
    else
        newTcaNode(NoConstraint());
        Synchro(t.labelPort);
        newOutTransition(t.actions);
```

execute the initial actions. If neither tpc nor timing constraint are defined in the initial automata, a node with no constraint is instantiated. Its out transition executes the initial actions.

NewTcaNode() and newOutTransition() functions and different considered cases answer to the first subtlety of the transformation. The use of the variable d_{ref} and update(d_{ref}) function goes with solving the second subtlety.

The Synchro() function in algorithm 1, answers to the third issue. It is responsible of adding synchronization points depending on if the transition is triggered by a send, receive or an internal port. Mainly it instantiates after/before variables updating, the suitable advance node bearing a well defined label at the rhythm of a fine-grained clock. The computation of the relation between the task clock and the fine-grained clock as well as the constraint supported by each instantiated advance node are subject of ongoing work.

C. Approach to prove the transformation correctness

An essential point to the transformation correctness proof approach is that the semantics of an RT-BIP component is defined as a Labelled Transition System (LTS).

In order to prove this correctness, we follow the method displayed in Figure 5. In this method we (i) express the semantics of TCA models in terms of LTS, (ii) consider the transformation between LTSs instead of the transformation between models directly and (iii) we prove that this transformation is semantics preserving using weak bisimulation technique. Then the direct transformation between models is correct by construction. This correctness proof method

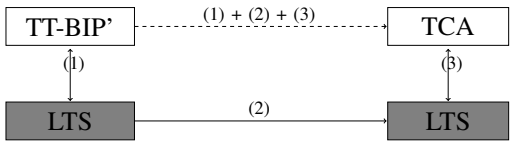

Fig. 5. Approach of the transformation correctness proof.

concerns correctness between an atomic component of TT-BIP model and its associated TCA model. However, to prove that the hole TT-BIP model is equivalent to the set of the obtained TCA models and their communication system calls, we need to prove that synchronization points preserve the same production/consumption order as in the initial model.

IV. Open challenges

We have identified several open challenges that we believe should be addressed when defining an optimized and generic transformation process.

Identification of OS service patterns potentially existing in the initial model: any OS has a number of services (communication, synchronization, etc.). We strongly think that in some initial models, and in components intended for handling communication, we can identify exactly the same behavioural pattern of one or more OS services. Transformation should take this redundancy into account, and only transform into TCA automata the part of the component which can not be mapped into an OS services. The identified pattern is thus mapped to a system call.

What about a generic transformation process? We strongly believe that the transformation process defined above can be generalized to any RTOS-based implementation approach with TT execution model. In fact, we just need to present the semantics of the computation model of the target platform as an LTS system.

V. Conclusion

In this paper we outline our approach to generate correct-by-construction TT implementation from high-level RT-BIP models. We divide this transformation into two steps; first we transform RT-BIP model in order to express intertask communication according to TT communication system, then we transform the obtained model into TCA automata (the computation model of PharOS applications). The first step is being done in previous work, we are now working on the second one. We define different subtleties induced by this transformation, and we give a short outline of the planned transformation strategy as well as the correctness proof process. We further identify open challenges related to our approach, that we plan to address with further research.

References

[1] Tesnim Abdellatif. *Rigourous Implementation of Real-Time Systems*. PhD thesis, UJF, 2012.

[2] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[3] C Aussagues, D Chabrol, V David, D Roux, N Willey, A Tournadre, and M Graniou. PharOS, a multicore OS ready for safety-related automotive systems: results and future prospects. *Proc. of The Embedded Real-Time Software and Systems (ERTS2)*, 2010.

[4] Simon Bliudze, Xavier Fornari, and Mathieu Jan. From model-based to real-time execution of safety-critical applications: Coupling SCADE with OASIS. In *Embedded Real Time Software and Systems*, ERTS2, page 10, February 2012.

[5] Borzoo Bonakdarpour, Marius Bozga, Mohamad Jaber, Jean Quilbeuf, and Joseph Sifakis. From high-level component-based models to distributed implementations. In *Proceedings of the tenth ACM international conference on Embedded software*, pages 209–218. ACM, 2010.

[6] Etienne Borde, Smail Rahmoun, Fabien Cadoret, Laurent Pautet, Frank Singhoff, and Pierre Dissaux. Architecture models refinement for fine grain timing analysis of embedded systems. In *Rapid System Prototyping (RSP), 2014 25th IEEE International Symposium on*, pages 44–50. IEEE, 2014.

[7] Jean-Louis Boulanger, François-Xavier Fornari, Jean-Louis Camus, and Bernard Dion. SCADE: Language and applications. 2015.

[8] Paraskevas Bourgos. *Rigorous Design Flow for Programming Manycore Platforms*. PhD thesis, Grenoble, 2013.

[9] Hela Guesmi, Belgacem Ben Hedia, Simon Bliudze, Saddek Bensalem, and Jacques Combaz. Towards time-triggered component-based system models. In *ICSEA15*, pages 157–169, Barcelone, Spain, November 2015. ThinkMind.

[10] Robert Kaiser and Stephan Wagner. Evolution of the pikeos microkernel. In *Proceedings of the 1st International Workshop on Microkernels for Embedded Systems*, pages 50–57, 2007.

[11] Hermann Kopetz. The time-triggered approach to real-time system design. *Predictably Dependable Computing Systems. Springer*, 1995.

[12] Matthieu Lemerre, Vincent David, Christophe Aussaguès, and Guy Vidal-Naquet. An introduction to time-constrained automata. In *Proc. of the* 3rd *Interaction and Concurrency Experience (ICE 2010)*, volume 38 of *EPTCS*, pages 83–98, 2010.

[13] Kathy Dang Nguyen, PS Thiagarajan, and Weng-Fai Wong. A UML-based design framework for time-triggered applications. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 39–48. IEEE, 2007.